ภาษาเครื่อง สำหรับภาษาสูง

ตอน *เขียนภาษาเครื่องของ 8086 บนไอบีเอ็ม พีซี*

บุญเลิศ เอี่ยมทัศนา*

ในตอนที่ผ่านมาได้กล่าวถึง ซีพียู จำลอง ชุดคำสั่งจำลองตลอดจนการเขียนโปรแกรมจำลอง เพื่อให้คุณผู้อ่านได้สัมผัสกับการเขียนโปรแกรมภาษาเครื่องในยุคแรก ๆ ของไมโครคอมพิวเตอร์บ้าง และได้รับทราบเรื่องราวบางเรื่อง อันจะเป็นพื้นฐานในการทำความเข้าใจต่อไป ซึ่งจะเป็นการนำของจริงมากล่าวถึง และผมจะขอนำเครื่อง IBM-PC/XT และภาษาปาสคาลมาเป็นแบบ (ผู้ที่ไม่ได้ใช้ภาษานี้ กรุณาติดตามด้วยครับ)

ภาษาเครื่องกับภาษาแอสเซมบลี

ได้มีผู้ข้องใจสอบถามว่า ภาษาเครื่องกับภาษาแอสเซมบลีต่างกันอย่างไร ? ขอเรียนให้ทราบว่าโดยส่วนรวมจะเรียกทั้งสองภาษานี้ว่าภาษาเครื่อง แต่ในการเขียนโปรแกรมภาษาเครื่องแท้ ๆ เราเขียนร่างโปรแกรมด้วยนิวมอร์นิกส์ในชุดคำสั่งเท่านั้น การตั้งชื่อใด ๆ เพื่ออ้างถึงเป็นเรื่องของแต่ละบุคคล ไม่ได้มีข้อกำหนดแต่อย่างไร การแปลเป็นรหัสเครื่อง (machine code) กระทำด้วยมือ

ส่วนการเขียนร่างโปรแกรมในภาษาแอสเซมบลี ซึ่งเมื่อป้อนเข้าเครื่องแล้วจะเรียกว่าซอร์สโปรแกรม (source program) นอกจากเขียนด้วยนิวมอร์นิกส์ในชุดคำสั่งแล้ว จะต้องเขียนตามข้อกำหนดของแอสเซมเบลอร์ (assembler) ด้วย ดังจะเห็นในบทความตอนนี้ การแปลเป็นรหัสเครื่องกระทำด้วยแอสเซมเบลอร์

แอสเซมเบลอร์ คือ โปรแกรมที่จะแปลซอร์สโปรแกรมเป็นรหัสเครื่อง แอสเซมเบลอร์บางตัว เช่น LISA ในเครื่องแอปเปิ้ลจะแปลซอร์สโปรแกรมเป็นรหัสเครื่องแท้ ๆ เช่นเดียวกับที่เราแปลด้วยมือ ซึ่งนำไปรันได้โดยตรง แอสเซมเบลอร์บางตัว เช่น MASM ซึ่งผมจะใช้ในบทความนี้ จะแปลซอร์สโปรแกรมเป็นรหัสเครื่องผสมกับรหัสจัดการในบางเรื่อง ทำให้ไม่สามารถรันได้ทันที ต้องใช้โปรแกรมอีกโปรแกรมหนึ่งคือ LINK มาดำเนินการก่อน จุดประสงค์เพื่อให้เชื่อมต่อโปรแกรมหลาย ๆ โปรแกรมเข้าด้วยกันได้ ซึ่งมีประโยชน์มาก

สำหรับแอสเซมเบลอร์ที่ใช้ประกอบบทความนี้จะใช้ MASM (Marcro Assembler) ของไมโครซอฟท์เวอร์ชั่น 1.0 ซึ่งแอสเซมเบลอร์ตัวนี้ ความจริงมีข้อบกพร่องอยู่บ้าง ที่ผมไม่ได้ใช้เวอร์ชั่น 4.0 เพราะเกรงว่าจะยังไม่เป็นที่แพร่หลาย

ภาษาเครื่องกับภาษาสูง

ข้อข้องใจอีกข้อหนึ่งคือ ที่ผมตั้งชื่อบทความชุดนี้ไว้ว่า "ภาษาเครื่องสำหรับภาษาสูง" มีความหมายอย่างไร ? เรื่องนี้สืบเนื่องมาจากบทความเรื่อง "เทอร์โบ–ปาสคาลกับภาษาเครื่อง" (ไมโครฯ 21) ได้มีผู้ร้องขอมาว่าผมเขียนบทความนั้นสำหรับผู้ที่รู้ภาษาเครื่อง ขอคำแนะนำบ้างสำหรับผู้ที่ไม่รู้ภาษาเครื่องมาก่อน

ผมได้พิจารณาดูแล้วเห็นว่าถ้าผมจะนำเรื่องราวของภาษาเครื่องทั้งหมดมากล่าวถึงจะเป็นเรื่องที่ยืดยาวมาก โดยเฉพาะการอธิบายถึงคำสั่งแต่ละคำสั่งในชุดคำสั่ง ถ้าผมตัดเรื่องราวบางเรื่องออกคงไว้เท่าที่จำเป็นในการสนับสนุนภาษาในระดับสูงเท่านั้น ยกตัวอย่างเช่น การคำนวณทำในภาษาสูงสะดวกอยู่แล้ว หากไม่นำมากล่าวจะทำให้บทความนี้สั้นลง ผมจึงให้ชื่อบทความชุดนี้ดังกล่าว เพื่อแสดงว่าไม่ใช่ภาษาเครื่องโดยสมบูรณ์ แต่ผมก็หวังว่าจะเป็นพื้นฐานอย่างดีสำหรับท่านที่สนใจศึกษาภาษาเครื่องต่อไป

จะกล่าวถึงอะไรบ้าง

ไม่มีท่านผู้ใดถามมาครับ แต่ผมเห็นว่าควรจะเรียนให้ทราบในโอกาสนี้

*รองหัวหน้าฝ่ายวิชาการ สถาบันวิชาการทหารเรือชั้นสูง

ด้วย คือผมจะกล่าวถึงซีพียู 8088 ในนามของ 8086 เพราะ 8088 คือ 8086 ที่มีขาดาต้าเพียง 8 ขา (8086 มี 16 ขา) และผมจะกล่าวถึงเฉพาะเรื่องราวของรีจิสเตอร์ชุดคำสั่ง กับแบบของการอ้างแอดเดรสเท่านั้น (คือไม่พูดถึงทางด้านฮาร์ดแวร์เลย) เรื่องต่อไปจะเป็นเรื่องของแอสเซมเบลอร์ MASM เฉพาะเท่าที่จำเป็น

สำหรับการเชื่อมต่อกับภาษาสูงนั้น ในตัวอย่างจะเป็นการใช้งานกับเทอร์โบ–ปาสคาลซึ่งเป็นการเรียกใช้โปรแกรมย่อย โปรดอ่านเรื่อง "เทอร์โบปาสคาลกับภาษาเครื่อง" (ไมโครฯ 21) ด้วยครับ เพราะผมจะนำมาใช้โดยไม่อธิบายอีก ส่วนท่านที่ใช้ภาษาซีขอให้ดูในหนังสือ "โปรแกรมคอมพิวเตอร์ภาษาซี" บทที่ 14 สำหรับภาษาเบสิกจะพยายามหาตัวอย่างมาเสนอในโอกาสต่อไป

ในเรื่องของเนื้อหาเมื่อกล่าวถึงเรื่องใดจะแสดงตัวอย่างประกอบไปจนจบบทความชุดนี้ ซึ่งจะเป็นตัวอย่างง่าย ๆ เพื่อประกอบความเข้าใจเท่านั้น อาจจะทำให้เห็นว่าไม่มีความจำเป็นใดในการใช้ภาษาเครื่องจะเห็นความจำเป็นต่อเมื่อนำโครงงานที่ใช้งานได้จริงมาเสนอ ซึ่งผมจะเขียนต่อจากบทความชุดนี้ตามโอกาสเป็นเรื่อง ๆ ไป และส่วนที่ผมเคยกล่าวไว้ว่าจะเขียนเรื่อง "ลึกลงไปในไอบีเอ็ม พีซี" ของคครับ เพราะทาง บ.ซีเอ็ดฯ กำลังจัดพิมพ์หนังสือที่มีเนื้อหาเดียวกันอยู่แล้วในขณะนี้

อนึ่ง ในการเขียนบทความนี้ ถึงแม้จะกล่าวว่าเป็นภาษาเครื่องสำหรับภาษาสูง แต่ในตัวอย่างส่วนมากจะเป็นภาษาเครื่องล้วน ๆ โดยไม่เกี่ยวกับภาษาสูง ทั้งนี้ก็เพื่อลดขั้นตอนในการเขียนโปรแกรมเพื่อทดลองรัน

รีจิสเตอร์

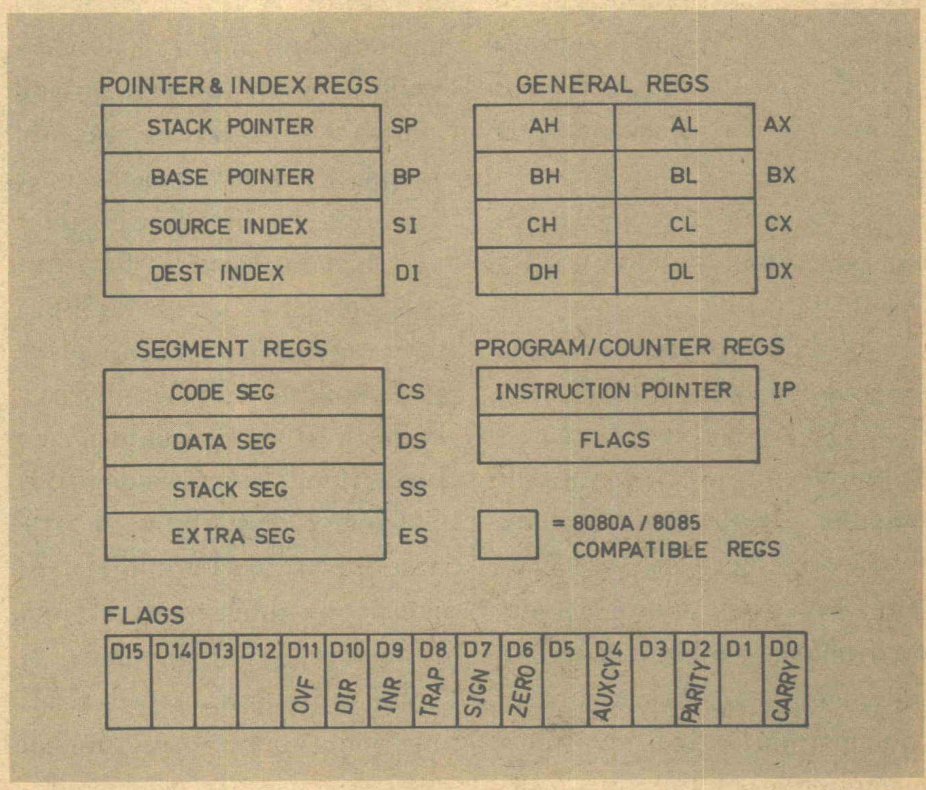
เป็นที่น่าเสียดายว่า 8086 มีแต่รีจิสเตอร์ขนาด 16 บิทเท่านั้น ทำให้มีเรื่องยุ่งยากบ้างในการอ้างแอดเดรส และทำให้ 8086 เป็นซีพียูขนาด 16 บิทโดยแท้ ไม่ใช่ 16 บิทบวกดังเช่น 68000 เพราะมีรีจิสเตอร์ขนาด 32 บิท รีจิสเตอร์ของ 8086 มีอยู่ 14 ตัว ดังแสดงในรูปที่ 1 ซึ่งสามารถแบ่งออกได้เป็น 6 กลุ่มดังนี้

1. รีจิสเตอร์ใช้งานทั่วไป (General purpose register) ประกอบด้วยรีจิสเตอร์ AX, BX, CX และ DX ซึ่งแต่ละตัวสามารถแยกเป็นรีจิสเตอร์ขนาด 8 บิทได้ 2 ตัว (เพื่อดำเนินกรรมวิธีกับข้อมูลขนาด 8 บิท) เรียกว่า High กับ Low เช่น AX แยกเป็น AH กับ AL (ขอให้ดูรูปที่ 1 ประกอบ) รีจิสเตอร์ทั้ง 4 ตัวนี้สามารถทำหน้าที่เป็นแอคคิวมูเลเตอร์ได้ทั้งหมด (ในซีพียูจำลองของผมมีแอคคิว–มูเลเตอร์เพียงตัวเดียว) และมีหน้าที่พิเศษอีกดังนี้

- AX เป็นแอคคิวมูเลเตอร์หลัก (Primary accumulator)
- BX เป็นเบสรีจิสเตอร์ใช้งานร่วมกับอินเด็กซ์รีจิสเตอร์ซึ่งจะกล่าวต่อไป
- CX ทำหน้าที่เป็นตัวนับในคำสั่งเกี่ยวกับการทำวงรอบ (loap)
- DX ใช้สำหรับเก็บแอดเดรสของพอร์ท คือใช้ในคำสั่งอินพุทและเอาท์พุท ว่าเราจะติดต่อกับพอร์ทใด

2. พอยน์เตอร์ (Pointer registers) มีอยู่ 2 ตัวคือ

- SP คือสแต็คพอยน์เตอร์ (Stack Pointer) ทำหน้าที่ชี้ชั้นบนสุดของสแต็ค ดังเช่น ซีพียูจำลองที่กล่าวมา
- BP คือเบสพอยน์เตอร์ (Base Pointer) ชื่อนี้ทำให้ฉงนมากสำหรับผู้ที่รู้จักซีพียูเบอร์อื่นมาก่อน เพราะชวนให้คิดว่าเป็นเบส คือฐานสำหรับรีจิสเตอร์ที่เป็นอินเด็กซ์เพื่อชี้ข้อมูลได้ทั่วไป ความจริงแล้ว มันเป็นฐานสำหรับอินเด็กซ์เหมือนกัน แต่จุดประสงค์เพื่อใช้ชี้ข้อมูลที่อยู่ในสแต็ค ซึ่งเราจะใช้รีจิสเตอร์ตัวนี้บ่อยพอสมควรเมื่อมีการผ่านค่าทางสแต็ค



รูปที่ 1 รีจิสเตอร์ภายใน 8086

3. อินเด็กซ์ (Index register) มีอยู่ 2 ตัว คือ SI กับ DI ทำหน้าที่ชี้ข้อมูลทั้งโดยตรง (ดังเช่น HL ในคอมพิวเตอร์จำลอง) หรือบนฐานคือ บน BX และ BP หากมีการย้ายข้อมูลระหว่างหน่วยความจำ 2 หน่วย โดยเฉพาะเมื่อย้ายข้อมูลเป็นจำนวนมาก จะแบ่งกันทำหน้าที่ดังนี้

- SI ชี้ต้นทาง (Source Index)
- DI ชี้ปลายทาง (Destination Index)

4. โปรแกรมเคาน์เตอร์ ทำหน้าที่ชี้หน่วยความจำที่เก็บคำสั่งดังเช่น ในซีพียูจำลอง แต่สำหรับ 8088 อินเทลกำหนดให้เรียกว่า IP (Instruction Pointer)

5. เซกเมนต์รีจิสเตอร์ โดยที่รีจิสเตอร์ต่าง ๆ ของ 8088 มีขนาดเพียง 16 บิท สามารถอ้างแอดเดรสได้เพียง 64 Kไบท์ แต่อินเทลออกแบบ 8088 ให้สามารถอ้างแอดเดรสได้ถึง 1 เมกะไบท์ ซึ่งต้องการรีจิสเตอร์ขนาด 20 บิท เพื่อเก็บค่าแอดเดรส อินเทลแก้ปัญหานี้ ด้วยจัดเซกเมนต์รีจิสเตอร์ขึ้นเพื่อใช้ช่วยในการอ้างแอดเดรส รายละเอียดจะขอนำไปกล่าวในหัวข้อการอ้างแอดเดรส ตอนนี้ขอให้ทราบชื่อไว้ก่อนมีอยู่ด้วยกัน 4 ตัว คือ

- CS คือโค้ดเซกเมนต์ (Code segment) เก็บค่าเซกเมนต์ของคำสั่งใช้ร่วมกับ IP
- DS คือ ดาต้าเซกเมนต์ (Data segment) เก็บค่าเซกเมนต์ของข้อมูล การกำหนด DS เราอาจกำหนดให้ทับกับ CS คือให้ค่าเดียวกันก็ได้ หรือจะกำหนดให้เหลื่อมกัน โดยที่เซกเมนต์หนึ่ง ๆ จะมีขนาด 64 Kไบท์ (ตามขนาดของรีจิสเตอร์) ดังนั้นถ้าเรากำหนดให้ CS และ DS ต่างกันเช่น 10 Kไบท์ เซกเมนต์ทั้ง 2 นี้ก็จะเหลื่อมทับกัน หรือเราจะกำหนดให้แยกออกจากกันได้ โดยกำหนดให้มีค่าต่างกันมากกว่า 64 Kไบท์ สำหรับในการเขียนโปรแกรมภาษาเครื่องสำหรับภาษาสูงส่วนมากจะกำหนดให้ทับกัน หรือใช้ที่กำหนดไว้ในภาษาสูง
- SS คือสแต็คเซกเมนต์ (Stack Segment) เก็บค่าเซกเมนต์ของสแต็ค ซึ่งในการเขียนโปรแกรมของเราจะไม่มีการกำหนด ปล่อยให้เป็นเรื่องของโปรแกรมตัวแปรภาษาสูงกับระบบจัดการ
- ES คือเอ็กซ์ตร้าเซกเมนต์ (Extra Segment) เก็บค่าเซกเมนต์ของข้อมูลเช่นกัน ใช้ร่วมกับอินเด็กซ์ เช่น ในการดำเนินกรรมวิธีกับข้อมูลที่เป็นสตริง โดยใช้ DS คู่กับ SI และ ES คู่กับ DI

6. สเตตัสรีจิสเตอร์ (Status register) หรือแฟลก (Flags) เป็นแฟลก (flag) 9 บิท อีก 7 บิทว่าง ดังแสดงในรูปที่ 1 ในตอนนี้ขอให้สนใจเฉพาะแฟลก C (Carry flag) กับแฟลก Z (Zero flag) ซึ่งจะมีการทำงานตรงตามที่กล่าวไว้ในซีพียูจำลอง (ไมโครฯ 23)

เลขฐานสิบหก

ถึงแม้การเขียนโปรแกรมภาษาแอสเซมบลีในปัจจุบันไม่จำเป็นต้องใช้เลขฐานสิบหก แต่การที่จะทำความเข้าใจกับแบบของการอ้างแอดเดรสที่จะกล่าวต่อไปจำเป็นต้องพึ่งเลขฐานสิบหกอยู่บ้าง ผมจึงขอนำมากล่าวถึง เพียงให้พอรู้เท่านั้น

ในระบบเลขฐานสิบเรามีตัวเลขอยู่ 10 ตัวคือ 0 ถึง 9 ในระบบเลขฐานสองมีตัวเลขอยู่ 2 ตัวคือ 0 กับ 1 ในระบบเลขฐานสิบหกมีตัวเลขอยู่ 16 ตัว คือ 0 ถึง 9 ต่อด้วย A ถึง F โดย A มีค่าเท่ากับ 10 ในเลขฐานสิบ ในการที่จะแสดงให้รู้ว่าเป็นตัวเลขของเลขฐานใดนั้น ในแอส-เซมเบลอร์ MASM ได้กำหนดไว้ว่า ถ้าเป็นเลขฐานสองให้เขียนอักษร B ต่อท้าย ถ้าเป็นเลขฐานสิบหกให้เขียนอักษร H ต่อท้าย แต่ถ้าขึ้นต้นด้วยเลขอักษรให้นำหน้าด้วยศูนย์ ส่วนเลขฐานสิบให้เขียนอักษร D ต่อท้าย หรือไม่ต้องเขียนก็ได้ และสำหรับเลขฐานสองนิยมเติมเลขศูนย์นำหน้าให้ครบไบท์ (8 บิทคือ 8 หลัก) หรือเวอร์ค (16 บิทคือ 16 หลัก) เช่น

ฐานสอง 0011 0101 B
1100 0011 0111 1111 B

ฐานสิบหก 35H
0C37FH

เหตุที่เราต้องใช้เลขฐานสิบหกเพราะว่าการเขียน-อ่านเลขฐานสองทำได้ยากแต่ผิดได้ง่าย จึงคิดจัดกลุ่ม เดิมจัดกลุ่มละสามคือ 3 บิท จะตรงกับเลขฐานแปดพอดี แต่ปัจจุบันไม่นิยมใช้ เปลี่ยนเป็นจัดกลุ่มสี่ คือ 4 บิท จะตรงกับเลขฐานสิบหกพอดี เช่น เลขฐานสองจำนวนแรกหากเขียนในแบบจัดกลุ่ม คือใช้ , คั่น จะเป็น 0011, 0101B ซึ่งเมื่อแปลงเป็นเลขฐานสิบหกจะได้เป็น 35H

สำหรับการแปลงเป็นเลขฐานสิบหกนั้น เนื่องจากในแต่ละกลุ่มมีเลขฐานสองอยู่เพียง 4 หลัก จึงทำได้โดยง่าย โดยจำไว้แต่เพียงว่า

n*8 + n*4 + n*2 + n*1

โดยที่ n คือค่า 1 หรือ 0 ของเลขฐานสอง หรือจำง่าย ๆ แต่เพียงว่า 8 4 2 1 ถ้าหลักใดเป็นเลข 1 ก็มีค่า เช่น 0011 จะได้เป็น 2+1 เท่ากับ 3 และ 1111 จะได้เป็น 8+4+2+1 = 15 คือ F

เรื่องนี้ต้องการการฝึกฝนครับจึงจะแปลงกลับไป-มาได้คล่อง ซึ่งยังไม่จำเป็นจะต้องสนใจนัก ขอให้จำให้ได้ว่าข้อมูล 1 ไบท์คือ 8 บิท จะเป็นเลขฐานสิบหก 2 หลัก ถ้า 2 ไบท์ที่ MASM เรียกว่าเวอร์ด (word) จะเป็นเลขฐานสิบหก 4 หลัก และที่พิเศษซึ่งจะเกี่ยวข้องกับการอ้างแอดเดรสคือ 20 บิทคือ 20 หลักของเลขฐานสองจะเป็นเลขฐานสิบหก 5 หลัก

การอ้างแอดเดรส

การอ้างแอดเดรสคือ การให้ค่าแอดเดรสแก่ขาแอดเดรสของซีพียู เพื่อติดต่อกับหน่วยความจำ ณ ตำบลที่หนึ่ง ๆ

ซึ่ง 8086 มีขาแอดเดรสอยู่ 20 ขา ทำให้สามารถอ้างแอดเดรสได้ถึง 2^{20} คือ 1 ล้านตำบลที่ ซึ่งต้องการค่าแอดเดรสถึง 20 หลัก หรือ 20 บิท ซึ่งต้องเก็บไว้ในรีจิสเตอร์ แต่ใน 8086 มีแต่รีจิสเตอร์ขนาด 16 บิท อินเทลแก้ปัญหานี้ด้วยการจัดเซกเมนต์รีจิสเตอร์ขึ้น 4 ตัวคือ CS, DS, SS และ ES เพื่อใช้ร่วมกับรีจิสเตอร์อื่น แต่ไม่ได้นำมาต่อกัน หากนำมาบวกกันในแบบเหลื่อมหลักของเลขฐานสิบหก เป็นค่าแอดเดรสจริง (EA : Effecting Address) เช่น

CS 9 F 0 1

IP 2 0 0 3

EA A 1 0 1 3

ในการอ้างแอดเดรสเราจึงต้องให้ค่าแอดเดรสทั้งในเซกเมนต์รีจิสเตอร์ และรีจิสเตอร์อื่น โดยจับคู่กันและนิยมเขียนกันดังนี้

CS : IP

SS : SP และ SS : BP

DS : BX และ DS : SI หรือ DI

ในการเขียนโปรแกรมภาษาเครื่องสำหรับภาษาสูง โดยเฉพาะกับปาสคาล ให้เรากำหนด CS ได้อย่างเดียว แต่ในตัวอย่างซึ่งเป็นภาษาเครื่องล้วน ๆ จะต้องกำหนด DS ด้วย ซึ่งผมจะกำหนดให้ทับกับ CS เมื่อกำหนดค่าเซกเมนต์แล้วเราไม่ต้องคำนึงถึงมันอีก MASM จะดำเนินการให้โดยอัตโนมัติ

รูปแบบของคำสั่ง อินเทลได้กำหนดรูปแบบของคำสั่งในชุดคำสั่งของ 8086 ไว้ 3 ลักษณะ เช่น

MOV dest, srce

INC dest/srce

CLC

คำสั่งแรกเป็นแบบที่มีโอเปอแรนด์ (operand) 2 ตัวคือ dest (อ่านว่า destination) กับ srce (อ่านว่า seurce) แยกจากกันด้วยเครื่องหมาย , ความหมายของคำสั่งในลักษณะนี้คือ ให้นำข้อมูลจาก srce คือต้นทาง มาดำเนินการกับ dest คือปลายทาง ซึ่งในกรณีนี้ให้ MOV คือย้ายข้อมูลจากต้นทางมาปลายทาง

คำสั่งที่สองมีโอเปอแรนด์เดียวที่เขียนเป็น dest/srce คืออาจจะมีลักษณะเป็นต้นทางหรือปลายทางก็ได้ ในกรณีนี้คือ INC มีลักษณะเป็นปลายทางคือ ให้เพิ่มค่า dest ขึ้นหนึ่ง ส่วนคำสั่งที่สามไม่มีโอเปอแรนด์ คำสั่งนี้ให้เคลียร์ค่าในแฟลก

แบบของการอ้างแอดเดรส

สำหรับ dest และ srce จะเป็นอะไรได้บ้างนั้นคือ แบบของการอ้างแอดเดรส ซึ่งถ้าแยกละเอียดจะมีหลายแบบมาก แต่สามารถสรุปโดยถือว่าคำสั่ง MOV คือคำสั่งที่มีแบบของการอ้างแอดเดรสทั้งหมด จะสามารถสรุปได้เป็น

MOV reg/mem, num

MOV mem, reg

MOV reg, reg/mem

โดยที่

- reg/mem หมายถึง reg หรือ mem ดังจะกล่าวต่อไป
- reg หมายถึง รีจิสเตอร์ขนาด 16 บิท คือ AX, BX, CX, DX, SI, DI, BP และ SP หรือรีจิสเตอร์ขนาด 8 บิท คือ AH, AL, BH, BL, CH, CL, DH และ DL คือเป็นการอ้างแอดเดรสในแบบ Register addressing เช่น

INC SI

MOV AX, DI

หมายถึงให้เพิ่มค่าใน SI ขึ้นหนึ่ง และนำค่าใน DI ให้แก่ AX

- num หมายถึง ข้อมูลขนาด 8 บิท หรือ 16 บิทที่อยู่ต่อท้ายคำสั่ง คือเป็น Immediate addressing เช่น

MOV AX, 2000H

หมายถึง ให้นำค่า 2000H ให้แก่ AX

- mem เป็นการอ้างแอดเดรสถึงหน่วยความจำในแบบ Direct หรือ Indexed addressing มีรูปแบบดังนี้

$$\left[\begin{Bmatrix}BX\\or\\BP\end{Bmatrix}+\begin{Bmatrix}SI\\or\\DI\end{Bmatrix}+\{disp/var\}\right]$$

เครื่องหมายวงเล็บปีกกา (เฉพาะในที่นี้) หมายถึง ว่าจะเลือกใช้หรือไม่ก็ได้ (แต่ต้องมีอย่างน้อยหนึ่ง) และสามารถเลือกต่อไปได้อีกเช่น เลือก BX หรือ BP เมื่อใช้ BP จะอ้างถึง SS นอกนั้นอ้างถึง DS เป็นเซกเมนต์รีจิสเตอร์ที่เขียนไว้ว่า disp/var นั้น คือ ถ้ามีแต่ค่านี้ค่าเดียวหมายถึง var (variable) เป็น direct addressing เช่น

MOV AX, [2000H]

หมายถึง ให้นำค่าในแอดเดรส DS : 2000H มาให้แก่ AX (ความหมายของ var คือ ตัวแปรจะเห็นชัดเจนเมื่อเขียนโปรแกรมแอสเซมบลี) แต่ถ้าใช้คำนี้ร่วมกับค่าอื่น เช่น SI จะเป็น disp (displacement) คือค่าขจัด เป็นการอ้างแอดเดรสในแบบ Indexed addressing เช่น

MOV SI, 3000H

MOV AX, [SI]

MOV BX, [SI + 1000H]

MOV AX, [BX + SI + 6]

คำสั่งแรกเป็นการให้ค่า 3000H แก่ SI คือ immediate addressing ที่กล่าวมาแล้ว คำสั่งที่สองเป็นการนำค่าในแอดเดรส DS : 3000H มาให้แก่ AX คำสั่งที่สามมาเป็นการนำค่าในแอดเดรส DS : 4000H มาให้แก่ BX สมมติว่า BX ได้ค่า 200H คำสั่งที่สี่จะเป็นการนำค่าในแอดเดรส 200H + 3000H + 6H คือ DS : 3206H มาให้แก่ AX จากที่กล่าวมานี้ไม่ทำให้เห็นว่า var กับ desp ต่างกัน แต่ขอให้สังเกตไว้ก่อนเพราะเมื่อเขียนโปรแกรมแอสเซมบลีจะแตกต่างกันอย่างมาก

รูปแบบของภาษาแอสเซมบลี MASM

ก่อนที่จะกล่าวถึงชุดคำสั่งของ 8086 ผมขอกล่าวถึงรูปแบบของภาษาแอสเซมบลี

```
;            PROGRAM  HELLO
; This program contain main program and procedure
;
cntr        EQU   10

data        SEGMENT                 ; Program Data Segment
letter      DB   "G"
string      DB   "ello world",13,10,"$"
data        ENDS


code        SEGMENT                 ; Program Code Segment
            ASSUME  CS:code, DS:data
start:                              ; Main program entry point
            mov  ax,data            ; set DS to data segment
            mov  ds,ax              ;  (register)
            mov  cx,cntr            ; number of count (immediate)
            inc  letter             ;  (direct)
time:       call hello              ; repeat 10 time
            loop time               ;  (relative)
            mov  si, OFFSET letter  ;  (immediate)
            sub  si, 50h            ;  (immediate)
            mov  dl, [si + 50h]     ;  (indexed)
            mov  ah,02h             ;  (immediate)
            int  21h
            mov  ah,4Ch             ; Return to DOS
            int  21h

hello       PROC NEAR               ; Procedure hello
            mov  dl,letter          ; get char (direct)
            mov  ah,02h             ;  call character display
            int  21h
            mov  dx,OFFSET string   ; load string (immediate)
            mov  ah,09h             ;  call string display
            int  21h
            ret
hello       ENDP

code        ENDS

stack       SEGMENT stack           ; Program Stack Segment
            DW  64 DUP (?)          ; Define stack space
stack       ENDS

            END
```

รูปที่ 2 ตัวอย่างโปรแกรมภาษาแอสเซมบลี ในวงเล็บคือคำอธิบายแสดงแบบของการอ้างแอดเดรสของ source ส่วน destination เป็น register ทั้งหมด

```
A>hello
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
H
A>
```

รูปที่ 3 ผลการรันโปรแกรมในรูปที่ 2

ของไมโครซอฟท์ คือ MASM (Macro Assembler) ก่อน ความหมายของคำว่า Macro เพื่อแสดงว่าเป็นแอสเซมเบลอร์ที่ให้สร้างส่วนของโปรแกรมให้เป็นมาโครได้ดังจะได้กล่าวในตอนต่อ ๆ ไป รูปแบบที่ผมนำมาแสดงนี้เป็นเพียงรูปแบบง่าย ๆ เท่านั้น ดังแสดงในรูปที่ 2 โปรแกรมนี้จะพิมพ์ข้อความว่า Hello Word ออกทางจอภาพ 10 บรรทัด และอักษร H อีก 1 ตัว ดังแสดงในรูปที่ 3

บรรทัด

โปรแกรมภาษาแอสเซมบลีประกอบด้วยคำสั่งต่าง ๆ คำสั่งละ 1 บรรทัด ซึ่งในแต่ละบรรทัดจะแบ่งออกเป็น 4 ฟิลด์ (field) แต่ไม่จำเป็นต้องมีครบทั้ง 4 ฟิลด์ อนึ่งตัวอักษรจะใช้ตัวเล็กหรือตัวใหญ่ จะมีความหมายเหมือนกันเพราะแอสเซมเบลอร์จะเปลี่ยนเป็นอักษรใหญ่หมด (เช่นเดียวกับในภาษาปาสคาล) ฟิลด์ทั้งสี่จะอยู่เรียงลำดับจากต้นบรรทัดดังนี้ คือ

– ฟิลด์ name เป็นการตั้งชื่อเพื่อเป็นชื่อของค่าคงตัว เช่น cntr, ตัวแปรเช่น letter, ชื่อของเซกเมนต์ เช่น data, ชื่อของเลเบิล เช่น start หรือเป็นชื่อของโพรซีเยอร์ เช่น hello

– ฟิลด์ action เป็นคำสั่ง (instruction) ของซีพียู เช่น MOV หรือเป็นไดเรคทีฟ (directive) ของแอสเซมเบลอร์ เช่น SEGMENT

– ฟิลด์ operand เป็นฟิลด์ที่ขยายความคำสั่ง เช่น ah, 4ch หรือขยายความไดเรคทีฟ เช่น NEAR

– ฟิลด์ comment ขึ้นต้นด้วย ; ข้อความที่ตามมาจนสุดบรรทัดคือข้อความใดก็ได้ซึ่งแอสเซมเบลอร์จะไม่สนใจ ทั้งนี้ก็เพื่อไว้เขียนคำอธิบายเพื่อให้ผู้อ่านโปร–

แกรมทำความเข้าใจได้ง่าย จะเขียนในแบบว่าต้องการให้เครื่องทำอะไร ไม่ใช่แสดงความหมายของคำสั่งนั้น ที่เขียนในรูปที่ 2 จุดประสงค์เพื่อแสดงแบบของการอ้างแอดเดรส

ฟิลด์ทั้งสี่นี้ แยกจากกันด้วยวรรค (space) จะกี่วรรคก็ได้ การขึ้นต้นบรรทัดจะวรรคหรือพิมพ์ชิดซ้ายก็ได้ โดยที่แอสเซมเบลอร์จะตรวจว่า

– ถ้าคำแรกปิดท้ายด้วย : เช่น start : และ time : จะถือว่าเป็นเลเบิลคือเป็นชื่ออ้างอิงถึงแอดเดรส ณ ตำแหน่งนั้น คือเมื่อเราเขียนโปรแกรมเราไม่ทราบว่าคำสั่งหนึ่ง ๆ จะอยู่ที่แอดเดรสใด จึงต้องใช้เลเบิลเป็นตัวกำหนด คำสั่ง loop time เป็นการกำหนดให้กระโดดกลับไปตรง time (และลดค่าใน CX ลงหนึ่ง จนกว่า CX = 0)

– ถ้าเป็นคำที่เป็นนิวโมนิคของ 8086 บรรทัดนั้นเป็นคำสั่ง

– ถ้าเป็นไดเรคทีฟ บรรทัดนั้นเป็นแอสเซมเบลอร์ไดเรคทีฟ ซึ่งในบทความนี้จะเขียนด้วยตัวใหญ่

– ถ้าไม่เข้าเกณฑ์ดังที่กล่าวมาจะถือว่าเป็นชื่อค่าคงตัวหรือตัวแปร หรือโปรแกรมย่อย

การกำหนดเซกเมนต์

เราต้องกำหนดเซกเมนต์ให้เป็นเซกเมนต์ของข้อมูล คำสั่ง และสแต็ค จะกำหนดอย่างละกี่เซกเมนต์ก็ได้หากต้องการเนื้อที่เกิน 64 K ไบท์ การกำหนดจะทำโดยกำหนดชื่อขึ้นแทนเซกเมนต์ก่อน เช่น data และ code ซึ่งขึ้นต้นด้วยไดเรคทีฟ SEGMENT และจบด้วย ENDS ภายในคือสิ่งที่กำหนดให้อยู่ในเซกเมนต์

การกำหนดเช่นนี้ยังไม่เป็นการบอกแอสเซมเบลอร์ว่า data และ code เป็นเซกเมนต์ของอะไร ชื่อที่เราตั้งขึ้นนี้เพียงเพื่อบอกผู้อ่านโปรแกรมว่าเราต้องการให้เป็นเซกเมนต์ของอะไร ซึ่งจะต้องกำหนดโดยใช้ ASSUME ดังในรูปที่ 2 จะเป็นการกำหนดว่า code เป็นโค้ดเซกเมนต์ และ data เป็นดาต้าเซกเมนต์

แต่ยังไม่ใช่เป็นการกำหนดค่าให้แก่ CS และ DS สำหรับ CS เป็นหน้าที่ของโปรแกรมจัดระบบงานคือ DOS เป็นผู้ให้ค่าเมื่อรัน ส่วน DS เราจะต้องให้ค่าเองดังที่เห็นในโปรแกรมโดยโหลดมาให้แก่ AX แล้วถ่ายค่าให้แก่ DS ในกรณีของสแต็ค การกำหนดดังที่เห็นในท่อนท้ายของโปรแกรม จะเป็นการกำหนดเซกเมนต์ชื่อ stack และให้เป็นสแต็คเซกเมนต์และให้ค่า SS ด้วย พร้อมทั้งกำหนดขนาดของสแต็คไว้ 64 ชั้น

ค่าคงตัวกับตัวแปร

การกำหนดค่าคงตัว เช่น cntr จะกำหนดไว้นอกเซกเมนต์ซึ่งอาจให้ค่าเป็นเลขหรืออักขระก็ได้ ค่าคงตัวนี้จะไม่ปรากฏในโปรแกรมที่เป็นรหัสเครื่อง ในโปรแกรมที่เขียนว่า mov cx, cntr จะถูกแอสเซมเบลอร์ตีความหมายว่าเป็น mov cx, 10

ต่างกับการกำหนดตัวแปร เช่น letter ซึ่งต้องกำหนดในดาต้าเซกเมนต์ เพราะจะเป็นการจองหรืออาจจะเรียกว่าเป็นการกำหนดหน่วยความจำขึ้น 1 หน่วย ในกรณีนี้ มีขนาด 1 ไบท์ เพราะใช้ DB (define byte) และให้มีค่าเป็น 47H คือ G ซึ่งตัวแปรนี้ในการใช้งานในโปรแกรมจะใช้เช่นเดียวกับรีจิสเตอร์ เช่น mov dl, letter จะมีลักษณะเหมือนกับ mov dl, al (มีข้อยกเว้นในการอ้างแอดเดรสแบบ indexed)

การกำหนดค่าให้แก่ตัวแปรอาจกำหนดโดยใช้ตัวเลข หรืออักษร หรือเป็นสตริงของตัวเลขหรืออักษร เช่น อักษร string ได้กำหนดให้เป็นสตริงของอักษรและตัวเลข ซึ่งตัวเลขที่ให้เป็นค่ารหัส ASCII ของ carriage return กับ line feed คือให้ขึ้นบรรทัดใหม่ สำหรับ $ เป็นข้อกำหนดของ DOS ให้ใช้ปิดท้ายสตริงเมื่อจะใช้โปรแกรมย่อยของ DOS

การกำหนดโปรแกรมย่อย

ได้กำหนดชื่อ hello เป็นชื่อของโปรแกรมย่อย จะเห็นว่าต้องขึ้นต้นด้วยคำว่า PROC และปิดด้วยคำว่า ENDP สำหรับคำว่า NEAR เป็นการกำหนดว่าเป็นโปรแกรมย่อยที่อยู่ในเซกเมนต์เดียวกันกับโปรแกรมที่เรียกมาคือตรง time ที่ call hello แอสเซมเบลอร์จะถือว่าเป็นการ call แบบ near คือเก็บเฉพาะ IP ลงสแต็ค และในการส่งกลับคือ ret ที่ท้ายโปรแกรมย่อย hello จะเป็นการ return แบบ near ด้วยคือจะถอนเฉพาะค่า IP คืนจากสแต็ค

หากเรากำหนดเป็นแบบ far คือเขียนว่า hello PROC FAR จะเป็นการ call แบบ far คือเก็บทั้ง CS และ IP ลงสแต็ค และจะเป็นการ return แบบ far คือถอนค่าทั้งสองนี้คืน ซึ่งจะไม่ใช้ในการเขียนโปรแกรมย่อยภาษาเครื่องสำหรับภาษาสูง โปรแกรมย่อยแบบ near นี้ ไม่จำเป็นต้องเขียนคำว่า NEAR คือถ้าไม่เขียนจะถือว่าเป็น near แต่ในบทความนี้จะเขียนเพื่อให้เข้าใจได้ง่าย

END

ท้ายสุดของโปรแกรมให้เขียนคำว่า END เพื่อบอกแอสเซมเบลอร์ว่าจบโปรแกรม

การทำงานของโปรแกรม

เริ่มตรง start ซึ่งจะเขียนบรรทัดเดียวกับ mov ax, data ก็ได้ สองบรรทัดแรกเป็นการโหลดค่า ดาต้าเซกเมนต์ให้แก่ ds (จะโหลดให้ ds โดยตรงไม่ได้) เสร็จแล้วให้ค่า 10 แก่ cx การเพิ่มค่าหนึ่งให้แก่ letter คือ inc letter ก็เพื่อแสดงให้เห็นว่าเราสามารถใช้ตัวแปรได้อย่างรีจิสเตอร์ ซึ่งจะเป็นการเพิ่มจาก G เป็น H ดังนั้นขณะนี้ตัวแปร letter มีค่าเป็น H คำสั่งต่อไปเป็นการเรียกใช้โปรแกรมย่อย Hello คำสั่ง loop time จะทำให้เรียกใช้ hello ซ้ำอยู่ 10 ครั้ง คือตามค่าที่ให้แก่ cx

ต่อไปเป็นการให้ค่าแอดเดรสของตัวแปร letter แก่ si จะเห็นว่าต้องใช้คำว่า

```
HELLO    ASM      1181  10-31-86   2:26a
        17 File(s)     73728 bytes free

A>masm hello
The IBM Personal Computer MACRO Assembler
Version 1.00 (C)Copyright IBM Corp 1981

Object filename [HELLO.OBJ]:
Source listing  [NUL.LST]:
Cross reference [NUL.CRF]:

Warning Severe
Errors  Errors
0       0

A>link hello

IBM Personal Computer Linker
Version 1.10 (C)Copyright IBM Corp 1982

Run File [HELLO.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

A>
```

รูปที่ 4 แสดงถึงวิธีการแอสเซมเบลอร์และลิงค์ ในทุกขั้นตอนที่เครื่องหยุดคอยให้กดคีย์ ENTER ซึ่งในการแอสเซมเบลอร์จะหมายถึงให้ไฟล์ .OBJ ชื่อ HELLO.OBJ ไม่สร้างไฟล์ .LST กับ .CRF ในขั้นตอนของลิงค์เป็นการเลือกไฟล์ .EXE ให้ชื่อ HELLO.EXE ไม่สร้างไฟล์ .MAP กับ .LIB

OFFSET หากใช้ว่า mov si, letter จะเป็นการนำค่า H มาให้แก่ si (และทำไม่ได้ด้วยเพราะเป็นไบท์เดียว) ที่ผมนำมาลบด้วย 50H ก็เพื่อจะแสดงการอ้างแอดเดรสในแบบ indexed addressing ในคำสั่งต่อไปเท่านั้น ซึ่งจะเป็นการโหลดค่าตรง letter ขณะนี้ คือ H มาให้แก่ dl คำสั่งต่อไปรวมทั้ง int 21 h เป็นการเรียกใช้ Function call ในดอส (ซึ่งต้องดูในคู่มือของดอส) ฟังก์ชั่นนี้เรียกว่าฟังก์ชั่น 02H ตามค่าที่ให้แก่ ah ซึ่งจะนำค่าอักขระใน dl ไปออกจอภาพ ต่อไปเป็นการใช้ฟังก์ชั่น 4CH เพื่อกลับไปดอส เป็นการจบโปรแกรม

ในโปรแกรมย่อย hello (ซึ่งทำงานก่อนนำค่า H ตัวเดียวไปออกจอภาพดังกล่าวมา) ได้โหลดค่าใน letter มาให้แก่ dl แล้วเรียกใช้ฟังก์ชั่น 02H เพื่อนำภาพ H ไปออกจอภาพเช่นกัน เสร็จแล้วโหลดค่าแอดเดรสต้นของตัวแปร string ให้แก่ dx แล้วเรียกใช้ฟังก์ชั่น 09H ซึ่งจะนำข้อมูลที่เป็นสตริงซึ่งมีแอดเดรสเริ่มต้นเป็น DS : DX และปิดท้ายสตริงด้วยอักขระ $ ไปออกจอภาพ ทำให้เกิดเป็นภาพ Hello World

ทบทวนการอ้างแอดเดรส

จากโปรแกรมตัวอย่างนี้ได้แสดงแบบของการอ้างแอดเดรสถึงข้อมูลแบบหลัก ๆ ทั้งหมดแล้วคือ

mov ds, ax เป็น register

sub si, 50H เป็น immediate

mov ax, data เป็น immdiate แบบพิเศษ

mov si, OFFSET letter เป็น immediate

mov dl, letter เป็น direct

mov dl, (si + 50h) เป็น indexed

แบบของการอ้างแอดเดรสดังกล่าวมานี้ หมายถึง source เท่านั้น ส่วน destisation เป็น register ทั้งหมด แบบของการอ้างแอดเดรสที่เหลือเป็นแบบปลีกย่อย

การเตรียมโปรแกรม

จะต้องทำเป็น 3 ขั้นตอน คือ

– ป้อนโปรแกรมนี้ ด้วยโปรแกรมประเภทเวิร์ดโปรเซสเซอร์ ซึ่งขอแนะนำให้ใช้ Note Pad ใน Sidekick จะสะดวกกว่าโปรแกรมตัวอื่นมาก

– ทำแอสเซมเบลอร์ โดย

A > masm hello

– ทำลิงค์โดย

A > link hello

ทั้งนี้โดยสมมติว่าไฟล์ทุกไฟล์อยู่ไดรฟ์ A หากอยู่ไดรฟ์อื่น ให้เขียนชื่อไดรฟ์นำหน้า เช่น A > masm B : hello การดำเนินการสองขั้นตอนหลังนี้ จะปรากฏภาพบนจอภาพดังรูปที่ 4 เมื่อดำเนินการเสร็จแล้วจะได้ไฟล์ชื่อ Hello.exe ซึ่งสามารถรันได้โดยตรง

ขอให้ทดลองดำเนินการครับ ฉบับต่อไปผมจะกล่าวถึงชุดคำสั่งของ 8086